# A New Coloured Petri Net Methodology for the Security Analysis of Cryptographic Protocols

Yongyuth Permpoontanalarp and Panupong Sornkhom
*Logic and Security Laboratory, Department of Computer Engineering*
*Faculty of Engineering*
*King Mongkut's University of Technology Thonburi, Bangkok, Thailand*
*E-mail: yongyuth.per@kmutt.ac.th*

**Abstract**

Cryptographic protocols are protocols which use cryptographic techniques to achieve certain tasks while preventing malicious parties to attack the protocols. The design and analysis of cryptographic protocols are difficult to achieve because of the increasingly attacking capabilities and the complex requirement of the applications. Therefore, attacks in many cryptographic protocols have been found later after they have been designed and even implemented. In this paper, we propose a new Coloured Petri Net methodology for the security analysis of cryptographic protocols. Our approach offers a simple but effective way to analyze multiple sessions of protocol execution. To demonstrate the practical uses of our approach, we apply our method to analyze Micali's contract signing protocol and TMN authenticated key exchanged protocol. Surprisingly we found many new attacks in those protocols.

## 1. Introduction

Cryptographic protocols are protocols which use cryptographic techniques to achieve certain tasks while preventing malicious parties to attack the protocols. There are many applications of cryptographic protocols, for example, authenticated key exchange protocols, web security protocols, e-payment protocols, e-banking protocols, e-voting protocols, etc.

The design and analysis of cryptographic protocols are difficult to achieve because of the increasingly attacking capabilities and the complex requirement of the applications. Therefore, attacks in many cryptographic protocols have been found later after they have been designed [1, 2, 3] and even implemented eg. [4, 5]. Note that in this paper we focus on only message replay attacks [6].

A lot of works on Petri nets [7] have been applied to analyze cryptographic protocols in [8-15]. They can be classified into two kinds. The first kind [8-14] offers a modeling and analysis method to find attacks. The second kind [15] provides a theoretical semantics for cryptographic protocols which can be used to prove properties of protocols, rather than to find attacks. We focus on the former kind due to its practical use. All the works in the first kind offer an analysis of a single session of protocol execution only.

In this paper, we propose a new Coloured Petri Net (CPN) methodology for the security analysis of cryptographic protocols. We adopt the CPN approach [16,17] due to the intuitive way to model cryptographic protocols by using the graph representation. Our approach offers a simple but effective way to analyze multiple sessions of protocol execution. Our new CPN methodology is based on our group's previous works [18-21]. We argue that our new CPN methodology improves on all existing CPN and Petri Net methods [8-14] for security protocols on several issues. In particular, our CPN method is the first CPN method which offers a security analysis methodology of multiple concurrent sessions of protocol execution. Furthermore, it offers a systematic method to analyze attacks in protocols. Also, it can detect more attacks with a better efficiency than all existing CPN methods for security protocols.

In essence, our new methodology offers four important concepts. Firstly, we use decomposition and multi-session scheduling techniques to analyze multiple sessions of protocol

execution. The decomposition allows us to construct a state space of one specific instance of multiple sessions of protocol execution at a time to reduce the size of the output state space. In general, such a specific instance means a setting or a configuration which specifies all required information for the execution of each concurrent session, for example, the parties who are involved in the protocol, all secrets, all nounces and all attackers in a session. The multi-session scheduling allows us to build a state space which contains only one alternating execution of multiple sessions of protocol run, instead of all possible alternating executions. As a result of these two techniques, the state space obtained is small and fast for analysis.

Secondly, we characterize attack states in the computed state space by using the concept of vulnerability events. Vulnerability events are events which may lead to a compromise of protocols, and such events are protocol dependent. The concept of vulnerability events provides a general method to characterize attack states intuitively and comprehensively. Thirdly, we develop an efficient method to extract attack traces without the need for any further computation on the state space. An attack trace describes how an attacker carries out their attacks successfully step by step. In other CPN works for security protocols, such an attack trace is computed by extracting a path from an initial state to an attack state. Fourthly, we propose a way to classify systematically a huge amount of found attack traces by using attack patterns. In general, an attack pattern describes the core of an attack, and it contains a minimal attack trace which leads to the attack. To the best of our knowledge, our four concepts are novel for the CPN methodology for analyzing cryptographic protocols.

To demonstrate the practical uses of our approach, we apply our methodology to two case studies which are Micali's contract signing protocol [22] and TMN authenticated key exchange protocol [23]. Surprisingly, we found many attacks in the two protocols. For Micali's contract signing protocol, we found new attacks in a single session of protocol execution in [19]. In [20], we also found many new multi-session attacks in both the original Micali's protocol and a modified version of Micali's protocol. For TMN authenticated key exchange protocol, in [21] we found six new attacks in the single session of protocol execution and two new attacks in multiple sessions of protocol execution. In fact, new attacks that we found in TMN protocol are quite surprisingly since TMN have been analyzed quite extensively [24-27].

In section 2, we provide the background on Micali's ECS1 and TMN protocol. In section 3, we compare our new CPN method with existing related works. In section 4, we present our new CPN methodology and apply it to two case studies which are Micali's ECS1 and TMN protocol.

## 2. Background

We use the following notations throughout the paper. $S \rightarrow R : M$ means that user $S$ sends message $M$ to user $R$. $SIG_X(M)$ represents party $X$'s signature on a message $M$ and we assume that $M$ is always retrievable from $SIG_X(M)$. The encryption of a message $M$ with party $X$'s public key is denoted by $ENC_X(M)$. Also, $H(C)$ stands for the hash of message $C$, and $E_K(M)$ means symmetric encryption on message $M$ by key $K$.

### 2.1. Micali's ECS1 Protocol [22]

Micali proposed an efficient optimistic fair exchange protocol for contract signing. The protocol aims to ensure that two exchanging parties get each other commitment on an agreed contract or neither of them does. There are three kinds of parties in the protocol : Alice as an initiator of the protocol, Bob as an responder of the protocol and a third trusted party who resolves a dispute between Alice and Bob during the exchange.

We denote Alice, Bob and a trusted party by A, B and TTP, respectively. It is assumed that both Alice and Bob have already agreed on a plaintext contract C before the exchange. Alice is committed to contract C as an initiator if Bob has both $SIG_A(C,Z)$ and M where $Z=ENC_{TTP}(A,B,M)$ and M is random. On the other hand, Bob is committed to C as a responder if Alice has both $SIG_B(C,Z)$ and $SIG_B(Z)$. However, there is no need for Alice to verify Z to prove Bob's commitment.

The following is the detail of the protocol.

A1: 1) $A \rightarrow B$: $SIG_A(C,Z)$

B1: 2) $B \rightarrow A$: $SIG_B(C,Z)$, $SIG_B(Z)$

A2: If Bob's signatures in step 2 are both valid, then

3) $A \rightarrow B$: M

B2: If Bob receives valid M such that $Z=ENC_{TTP}(A,B,M)$
then the exchange is completed
else Bob requests TTP to resolve a dispute by the following step
4) B→TTP: A, B, Z, $SIG_B(C,Z)$, $SIG_B(Z)$
TTP1: If Bob's signatures in step 4 are both valid and $Z=ENC_{TTP}(A,B,M)$ then
5a) TTP→A: $SIG_B(C,Z)$, $SIG_B(Z)$
5b) TTP→B: M

Note that the request to TTP at step 4) contains identities of initiator and responder which have the dispute. Also, to resolve the dispute, TTP sends required information to related parties.

In [28], Bao et al. analyzed ECS1 manually and found three message replay attacks in ECS1. It can be argued that many of these attacks are caused by an ambiguity at how TTP should resolve a dispute. Then, Bao et al. modify ECS1 to solve the ambiguity problem in the original ECS1, and they found one attack in the modified version. In other words, Bao showed that a simple modification on TTP's behavior to resolve a dispute is not adequate. The modified ECS1 is similar to the original ECS1 except that in step TTP1, there is no check on $Z=ENC_{TTP}(A,B,M)$ and messages in steps 5a) and 5b) are also sent to identities A and B which are in step 4 and are obtained from the decryption of Z. In other words, in the modified ECS1, TTP resolves the dispute even when Z is not correct, ie. $Z \neq E_{TTP}(A, B, M)$.

In [29], Zhang and Liu applied a model checking technique to analyze the security of ECS1. They found three new attacks. We will discuss about their new attacks and the comparison with our new attacks in section 3.2.

### 2.2. TMN authenticated key exchange protocol [23]

TMN is a cryptographic key exchange protocol for mobile communication system. TMN allows user A to exchange a session key with user B by the help of server J. The user A is called an initiator, but the user B is called a responder. The detail of TMN is described as follows.

1) $A \rightarrow J : (B, ENC_J(K_{aj})), A$
2) $J \rightarrow B : A$
3) $B \rightarrow J : (A, ENC_J(K_{ab})), B$
4) $J \rightarrow A : B, E_{Kaj}(K_{ab})$

Where $K_{ab}$ is an exchanged session key and $K_{aj}$ is A's secret which is used to transport the session key at the last step. Note that the session key is created by user B. In [23], it is suggested that the Vernam cipher (or one-time pad) and RSA public key algorithm are used as the underlying symmetric encryption $E_K(M)$ and the public key encryption, respectively.

TMN has been analyzed extensively by many formal method approaches [24-27], and many attacks were found. But it was analyzed manually also in [23] by Simmon. Simmon found an attack to TMN by using the homomorphic property of the underlying public key cryptographic algorithm. An attacker can learn an exchanged session key easily, and the server cannot detect any message replay. The attack involves two concurrent sessions of protocol execution. We will discuss about the analysis of TMN by formal methods in section 3.2.

## 3. Related works

### 3.1. Petri Nets for Cryptographic Protocols

A lot of works on Petri nets [7] have been applied to analyze cryptographic protocols in [8-15]. They can be classified into two kinds. The first kind [8-14] offers a modeling and analysis method to find attacks. The second kind [15] provides a theoretical semantics for cryptographic protocols which can be used to prove properties of protocols, rather than to find attacks. We focus on the former kind due to its practical use.

All the works in the first kind [8-14] offer an analysis of a single session of protocol execution only. Moreover, the works in [8-11] employ some extended Petri nets to analyze security protocols but those Petri nets are less expressive than CPN. In [12], CPN is applied to analyze a denial of service attack on cryptographic protocols in a single session of protocol execution. The work [12] analyzes protocols by using the simulation technique but our work analyzes protocols by

using the state space computation. Furthermore, while their work analyzes the denial of service attack, we analyze attacks on the confidentiality, the authentication, key exchange, fair exchange properties. So the kinds of attacks are different.

There are two works [13, 14] which are closest to ours. They apply CPN to analyze cryptographic protocols on similar kinds of attacks to ours. However, both works do not really provide an analysis of multiple concurrent sessions of protocol execution, but just two sequential sessions of protocol execution. Even though [13] offers an analysis of two sequential sessions of protocol execution, it has to analyze one session at a time. In other words, the analysis of two sequential sessions is non-compositional. In particular, after an execution of the first session is finished, the second session is then executed separately with information obtained from the first session. So, it is considered as a single session analysis strictly. However, the work [14] offers a compositional analysis of two sequential sessions of protocol execution according to the detected attack reported in the work. The analysis is compositional in that two sequential sessions can be analyzed at a time. Since it is a sequential composition of two sessions, then the work is essentially a single session analysis. Moreover, the attacker cannot initiate a new session with any user. So, only attacks where a legitimate user is a session initiator can be detected, and these are very limited. More importantly, a methodology to deal with a huge amount of alternating executions between multiple sessions has not been addressed at all in both works. But the work [14] deals with the alternating executions between a user and an attacker, since their attacker model can be executed independently of the execution of protocol steps. Such alternating executions occur in a single session only.

It is important to note that the analysis of multiple concurrent sessions of protocol execution is important since many crucial attacks, for example the man-in-middle attack [30] and parallel-session attack [31] are carried out in the multiple concurrent sessions where protocol runs are alternated in a non-sequential manner between multiple sessions.

Recently in [18-21], our group has developed a new CPN method to analyze cryptographic protocols. Our new CPN method improves on all existing CPN and Petri Net methods for security protocols on several issues. In particular, our CPN method is the first CPN method which offers an analysis methodology of multiple concurrent sessions of protocol execution. Furthermore, it offers a systematic method to analyze attacks in protocols. In particular, our method offers decomposition and multi-session scheduling for the computation of state spaces, an intuitive approach to characterize attack states, an efficient way to extract attack traces and a systematic way to classify a large amount of attack traces. In addition, our method can detect more attacks with a better efficiency. In particular, our attacker model allows session initiation, receiver impersonation and message dropping capabilities all of which are missing in [14]. As a result, more kinds of attacks can be analyzed. Also, our method is more efficient due to more restricted underlying models of users and attackers. In particular, in the user model there is a strict message validation for every received message, and each type of cryptographic messages, eg. public-key ciphertext and symmetric-key ciphertext, is defined individually instead of mixing them into one type Also, our attack model generates messages by taking both the message validation and the individual types of cryptographic messages into account. As a result, a state space generated is smaller than [14] since invalid messages and inappropriate cryptographic messages are eliminated.

### 3.2. Other formal methods for analyzing Cryptographic protocols

There are a huge amount of formal methods for analyzing cryptographic protocols. A survey and discussion on the comparison between them can be found in [2,3]. Here we discuss only formal methods which are applied to analyze contract signing protocols, ECS1 and TMN protocols.

There are at least three main works which analyze general contract-signing protocols. In [32], Chadha, Kanovich, and Scedrov proposed an inductive proof method to analyze a variant of contract-signing protocol proposed by Garay, Jakobsson and MacKenzie. Their method aims to prove fairness and abuse-free properties of the protocol, rather than to find attacks. In addition, their method is manual. In [33], Shamatikov and Mitchell applied a model checking system called Mur$\varphi$ to analyze two contract signing protocols. A protocol is modeled as an automata by using a programming language. Their method is automatic and some new attacks on the protocols are discovered. In [34], Gurgens and Rudolph analyzed a number of fair exchange non-repudiation protocols using asynchronous product automata (APA) and the simple homomorphism verification

tool (SHVT). Similar to Mur$\varphi$, a protocol is modeled as an automata but by using a text-based description on states and state transitions. Their method is automatic and some new attacks are found.

In [29], Zhang and Liu applied a model checking technique to analyze ECS1. They found one new single-session attack in Micali's ECS1 and two new multi-session attacks in Bao et. al. 's modified version of ECS1 [28]. Independently, we found two new single-session attacks in Micali's ECS1 in [19], and found two new multi-session attacks in Micali's ECS1 and five new multi-session attacks in Bao's modified version of ECS1 in [20]. In fact, at the writing time of our works in [19, 20], we were unaware of Zhang and Liu's work. However, after we look into the details, we found that one of our single session attacks is a variant form of Zhang and Liu's attack, and two of our multi-session attacks are variant forms of Zhang and Liu's attack. Therefore, to summarize the new attacks found by our CPN method we found one new single-session attack of Micali's ECS1, two new multi-session attacks in Micali's ECS1 and three new attacks of Bao's modified version of ECS1.

There are seven formal method approaches which are applied to analyze TMN protocol. In [24], three formal method approaches, namely NRL, Interrogator and Inatest, to analyze cryptographic protocols are compared, and the TMN protocol is chosen as a case study. All of them take the state exploration approach to analyze protocols. Both NRL and Interrogator detect an attack in a single session of protocol execution. However, Inatest can only reproduce Simmon's attack [23] which is an attack in multiple sessions. In [25], Mur$\varphi$, a general model checker, is applied to analyze the TMN protocol. Mur$\varphi$ can reproduce Simmon's attack, and it detects a new multiple session attack which allows an attacker to learn an exchanged session key between two legitimate parties. However, both legitimate parties do not commit on the session key after the completion of the protocol. In [26], Lowe and Roscoe applied Communicating Sequential Processes (CSP) and its model checker FDR to analyze the TMN protocol. Not only all previously known attacks to the TMN protocol can be reproduced, but also two new attacks have been found. The first new attack occurs in a single session of protocol execution, and an attacker can impersonate user A and learn the exchanged session key created by user B. The second attack which occurs in multiple sessions is that both A and B commit on the same session key after the completion of the protocol, but the key is known by the attacker. In [14], Al-Azzoni et. al. applied CPN to analyze the TMN protocol. Their CPN method can only detect a variant form of the attack found by Mur$\varphi$ [25]. Note that the attack occurs in two sequential sessions of protocol run. In [27], Zhang and Liu employed a model checking technique to analyze the TMN protocol. They found some variant forms of Lowe and Roscoe's attacks [12] in both a single session and multiple sessions. Even though TMN protocol has been analyzed very extensively, surprisingly we found new attacks in the protocol by using our CPN methodology.

In summary, we argue that the CPN methodology offers an advantage over those formal methods discussed previously in that it provides a simpler and more intuitive way to model a protocol by using the graph representation.

## 4. Our Model

### 4.1. Our New Methodology

Our new CPN method here is based on the CPN approaches that we have developed earlier in [18-21]. There are five steps. The detail of our methodology is shown as follows.

First, we build a CPN model to represent message exchange by all user parties and to represent attacker behavior. Each user party is modeled according to the protocol. In general, an attacker in our model can eavesdrop, modify and drop messages during the transmission. Also, the attacker can send new messages. We will discuss about assumptions of the protocol and the abilities of the attacker in details later.

Second, an automata or a state space of the protocol with attackers is generated by using the state space tool in CPN Tools [17]. In general, the state space represents all possible behaviors of every party, including attacker, in the protocol. Instead of generating the state space of all possible instances of multiple concurrent sessions of protocol execution at once, we generate a state space of one specific instance of multiple concurrent sessions at a time to reduce the size of the output state space. We call this a decomposition technique. In general, such a specific instance means a setting or a configuration which specifies all minimally required information for the protocol execution, for

example, the identities of initiator and responder, the role of attackers, all secrets and all nounces in each concurrent session, and a schedule of the execution of the multiple concurrent sessions. The schedule ensures that the output state space contains one alternating execution of multiple concurrent sessions of protocol run only, instead of all possible alternating executions. Exploring all possible alternating executions within a state space is expensive and causes a huge state space. Usually there are many possible configurations. As a result of the decomposition and the multi-session scheduling techniques, the state space obtained is small and fast for analysis.

Third, we create a query function in CPNML language, which is based on ML functional programming language, to search for attack states in the state space. Attack states are characterized by vulnerability events. Vulnerability events are events which may lead to a compromise of protocols, and such events are protocol dependent. In ECS1, there is one vulnerability event where one party, who is either initiator or responder, gets another party commitment, but the latter does not get the former commitment. In other words, this event describes exactly an unfair state. In TMN protocol, there are two main vulnerability events. Firstly, the attacker learns a secret key, for example, the exchanged session key. Secondly, a session key which may be a fake key is committed by a user. Based on these two vulnerability events, several combined vulnerability events are created in order to characterize many meaningful attack states in TMN protocol. The concept of vulnerability events provides a general method to characterize attack states intuitively and comprehensively. Also, queries can be built easily to detect such combined vulnerability events.

Fourth, after attack states are discovered from the state space, we extract attack traces. Conceptually, an attack trace describes how an attacker carries out an attack successfully step by step. Given an attack state in a state space, a path from the initial state to the attack state contains a sequence of actions by all parties, including attackers, which leads to the attack. So, this sequence contains an attack trace. But the sequence contains superfluous and redundant information about an attack trace since it contains all CPN transitions in an execution which lead to an attack state, and many of those transitions are redundant or irrelevant to the essence of the attack. Our method offers an efficient new approach to extract attack traces from an output state space without the need for any further computation. In our CPN model, as the protocol execution proceeds, an attack trace which is a record of all exchanged messages between parties so far is embedded into an output state. More specifically, the attack trace is stored in a global fusion place when a message is sent from one user to another. Thus, when an attack state is found, the attack trace can be extracted from the state immediately and efficiently.

Fifth, after attack traces are obtained, we classify them into each group. In general, there can be a huge amount of attack states and traces found in a state space. For example, in ECS1 we found 7,000 attack states (and traces) in a configuration. Thus, to ease the analysis of a large amount of attack traces, we develop an attack classification by using attack patterns. In general, an attack pattern describes the core of an attack, and it contains a list of minimal protocol messages that are the cause of each attack. Attack traces that produce the same attack pattern are classified into the same group of attacks. In general, it requires human intervention to create each attack pattern from an attack trace.

In the next two sections, we apply our CPN method to analyze two case studies. To illustrate the effectiveness of our method, we focus our analysis on TMN. Our CPN framework for ECS1 is similar to the framework for TMN.

### 4.2. Our CPN Analysis for TMN Protocol

In this section, we discuss the analysis of TMN by using our new methodology. In section 4.2.1, we provide definitions and more concrete concepts of our methodology. In section 4.2.2, we briefly show some of our CPN graph model. We explain our queries and our attack classification technique in sections 4.2.3 and 4.2.5, respectively. Also, new attacks and the performance of our method are discussed in sections 4.2.6 and 4.2.7, respectively.

#### 4.2.1. Our CPN framework for TMN

In this section, we discuss the assumptions of our protocol analysis. We also describe vulnerability events of TMN. Finally, we provide a definition of a configuration of the protocol execution.

**Definition 1** : The assumptions of the protocol execution
The following are the assumptions of the execution of the TMN protocol.

1. There are three users who are an initiator, a responder and a server. And all the users follow the protocol specification strictly and honestly.
2. There is one attacker whose abilities are defined below.
3. The underlying encryption is perfect in that nothing can be inferred from a ciphertext without the knowledge of the correct key. Also, we consider a general public key encryption scheme, rather than any specific scheme.
4. We consider the execution of two concurrent sessions of the protocol where such execution can be performed in either a sequential or a non-sequential but alternating style.
5. Both initiator and responder involve in the protocol execution as if there is one session of execution only, but the server may involve in more than one session.

The assumptions 1) and 2) mean that those parties are all that are involved in the protocol execution. An initiator means a user who initiates a new protocol session, and a responder means a user who responds to an existing session with an initiator to perform the key exchange.

The first part of assumption 3) is also known as Dolev and Yao's assumption [35]. As stated in the second part of assumption 3), in this paper we consider a general public key encryption scheme rather than RSA scheme. This assumption is sufficient to illustrate the effectiveness of our new CPN approach to analyze TMN.

In assumption 4), we mean that in addition to a sequential execution of two sessions, the protocol execution can alternate between the two concurrent sessions. Thus, the result of the execution is a non-sequential or interleaving manner. We will discuss about the execution of two concurrent sessions of the protocol more specifically later.

In assumption 5), we mean that both initiator and responder think that they involve in only one session of the protocol execution, and their goals are to exchange a session key between them. However, the server may involve in more than one session of the execution. In fact, this assumption is reasonable since a server just responds to any request and it maintains a state only during a request at step 1 and a respond at step 4. After the step 4 occurs, the state is destroyed to minimize the resource.

**Definition 2** : The attacker abilities
The attacker in our model is capable of the following:

1. The attacker can eavesdrop, modify and drop messages during the transmission between users.
2. The attacker can send a message to a user.
3. The attacker can either initiate a new session with users or take part in an existing session with users.
4. The attacker can impersonate any user.
5. The attacker can perform any cryptographic computation by using known keys, known messages and known ciphertexts with a limited but reasonable power, eg. encryption and decryption.
6. The attacker has its own storage with a finite and reasonable amount.
7. The attacker does not attack himself.
8. There is at most one attacker who performs the attack in 1) on a protocol step in a session at a time.

Note that 2) means the ability to send any message to any user where the message and the user may or may not be according to the protocol specification. Thus, it is different from 3).

The assumption 4) means that the attacker can impersonate an initiator A, a responder B or a server J. If the impersonated user is a responder or a server, then the attacker must be able to intercept an input message and then to send a fake respond message at a next step. But if the impersonated user is an initiator, the attacker must be able to initiate a new session, and to generate fake responds.

In 5), the attacker also has the ability to perform any computation in addition to the cryptographic one.

In 7), any message that is sent to an attacker who may impersonate a user will not be modified by anyone during the transmission. Note that if there is any modification, then the attacker is attacked. In 8), any message that is sent from an attacker will be delivered to the intended receiver

intact. If there is anyone else who modifies the sent message further, then the message is modified by two attackers.

**Definition 3** : The basic goals of the attacker
There are two basic goals of the attacker for TMN
1. The attacker aims to disclose a secret key which is a session key or A's secret.
2. The attacker aims to impersonate a user which is an initiator or a responder

These are two basic and general goals of the attacker for TMN. We focus on the first goal, but still consider the second goal, since the first goal incurs a worse damage than the second one. In addition, the attacker has more aims to compromise the protocol by achieving the combined vulnerability events which will be discussed later. Those events can be considered as concrete and advanced goals of the attacker.

Attack states are characterized by vulnerability events. For the TMN protocol, there are two basic events, and five combined events.

**Definition 4** : The first basic vulnerability events
The attacker learns a secret key. There are two cases.
1. The attacker learns the exchanged session key $K_{ab}$. *(K_$K_{ab}$)*
2. The attacker learns A's secret $K_{aj}$. *(K_$K_{aj}$)*

**Definition 5** : The second basic vulnerability events
There are three cases for a session key which is committed by users.
1. Both A and B commit on $K_{ab}$. *(AB_$K_{ab}$)*
2. A commits on $K_i$ or $K_{aj}$. *(A_$K_i$, A_$K_{aj}$)*
   where $K_i$ is attacker's secret key.
3. B commits on $K_{ab}$. *(B_$K_{ab}$)*

Note that 1) does not look like a vulnerability event on its own, but when it is combined with another vulnerability event, it becomes a vulnerability event clearly. We will discuss about this later. Also, it is not possible to fool user B to commit to other key than $K_{ab}$ since B is the creator of the session key $K_{ab}$ and then the key is fixed for the communication with A. Based on the two basic events, the following combined and interesting vulnerability events can be created.

**Definition 6** : The combined and interesting vulnerability events.
There are five combined vulnerability events.
1. The attacker learns $K_{ab}$, and both A and B commit on $K_{ab}$. *[$K_{ab}$][$K_{ab}$][$K_{ab}$]*
2. The attacker learns $K_{ab}$ and $K_{aj}$, and both A and B commit on $K_{ab}$. *[$K_{ab}$,$K_{aj}$][$K_{ab}$][$K_{ab}$]*
3. The attacker learns $K_{ab}$, and A is fooled to commit on $K_i$ but B commits on $K_{ab}$. *[$K_{ab}$][$K_i$][$K_{ab}$]*
4. The attacker learns $K_{ab}$ and $K_{aj}$, and A is fooled to commit on $K_i$ but B commits on $K_{ab}$. *[$K_{ab}$,$K_{aj}$][$K_i$][$K_{ab}$]*
5. The attacker learns $K_{ab}$ and $K_{aj}$, and A is fooled to commit on $K_{aj}$ but B commits on $K_{ab}$. *[$K_{ab}$,$K_{aj}$][$K_{aj}$][$K_{ab}$]*

We use the notation *[$KB_1$][$KB_2$][$KB_3$]* to describe each combined vulnerability event where $KB_1$ stands for keys that are known by the attacker, and $KB_2$ and $KB_3$ stands for and keys that are committed by users A and B, respectively, at the completion of the protocol.

Clearly, in the event 1 the attacker will then learn all later communication between A and B, because the attacker obtains the session key which are exchanged and agreed by both A and B. We do not find any attack instance in this event.

The event 2 is similar to the event 1 but the attacker in the event 2 learns an additional key which is A's secret. Lowe and Roscoe's multi-session attack [26] is in the category of this event. In this event, we found 10 attack patterns, and many of them are interesting variant forms of Lowe and Roscoe's attack. For example, in some variant attacks, server J cannot detect the replay attack occurred in the two sessions. Its detail will be discussed later.

In event 3, the attacker learns the session key, and fools A to commit to a fake key which is the attacker's secret $K_i$. The event 4 is similar to the event 3, but the attacker learns A's secret key in

addition the session key. In event 5, the attacker learns the session key and A's secret key, and fools A to commit to a fake key which is A's secret key $K_{aj}$. The result of the events 3, 4 and 5 can be seen as a kind of the man-in-the-middle attacks. In events 3 and 4, the attacker can impersonate B to A by using key $K_i$, while the attacker can impersonate A to B by using key $K_{ab}$. In the event 5, the attacker can impersonate B to A by using key $K_{aj}$, while the attacker can impersonate A to B by using key $K_{ab}$. We found 10 attack patterns in the events 4 and 5, but do not find any attack instance in the event 3. It is the attacks in the events 4 and 5 that are novel.

Note that there are other two combined vulnerability events which are $[K_{aj}][K_i][K_{ab}]$ and $[K_{aj}][K_{aj}][K_{ab}]$. However, both events are not interesting since the attacker does not learn the session key $K_{ab}$. Moreover, they are just variant forms of single session attacks. So, we do not consider them here.

**Definition 7** : The computation of the five combined vulnerability events

The following shows how to compute the five combined vulnerability events.

1. $[K_{ab}][K_{ab}][K_{ab}] = (K\_K_{ab} \cap AB\_K_{ab})$
2. $[K_{ab},K_{aj}][K_{ab}][K_{ab}] = (K\_K_{ab} \cap A\_K_i \cap B\_K_{ab})$
3. $[K_{ab}][K_i][K_{ab}] = (K\_K_{ab} \cap K\_K_{aj} \cap AB\_K_{ab})$
4. $[K_{ab},K_{aj}][K_i][K_{ab}] = (K\_K_{ab} \cap K\_K_{aj} \cap A\_K_i \cap B\_K_{ab})$
5. $[K_{ab},K_{aj}][K_{aj}][K_{ab}] = (K\_K_{ab} \cap K\_K_{aj} \cap A\_K_{aj} \cap B\_K_{ab})$

Each combined vulnerability event can be computed by applying the intersection on the relevant basic vulnerability events.

In the following, we provide the definition of a configuration for a state space computation of our CPN framework to analyze the TMN protocol.

**Definition 8** : A configuration of our CPN framework for TMN

A configuration of our CPN framework consists of $((S_1, S_2,\ldots,S_n), Sch, Tr)$ and $S_i = (s,I,R,T,K,N)$ for $1 \le i \le n$ where

1. $S_i$ is a session information which consists of
   - 1.1. $s$ is a session identity
   - 1.2. $I$ is an initiator identity
   - 1.3. $R$ is a responder identity
   - 1.4. $T$ is a server identity
   - 1.5. $K$ is keys for each party (including attacker) which consists of
     - a) Pair of public and private keys
     - b) Shared key with a specific party
   - 1.6. $N$ is nounces used by each party
2. $Sch$ is a multi-session schedule which contains a list of session identities to be executed in that order
3. $Tr$ is an attack trace which consists of a vulnerability event and a list of protocol traces which leads to the vulnerability event

In the configuration, each $S_i$ and $Sch$ are input parameters to the CPN state space computation while $Tr$ is the desired output from the state space computation.

For example, the 1st session information $S_1$ which is $(1,A,B,J,(K_1,K_2,K_3,K_4),\_)$ means that A, B and J are identities for initiator, responder and server, respectively, and $K_1$, $K_2$, $K_3$ and $K_4$ are keys for $A$, $B$, $J$ and $In$ respectively. Also, "_" means that there is no information about the nounces. Let $K_1 = (\_,\{(K_{aj},\_)\})$, $K_2 = (\_,\{(K_{ab},A)\})$, $K_3 = (\{(PK_J,SK_J)\},\_)$ and $K_4 = (\_,\{(K_i,\_)\})$. $K_1$ means that A has no public and private keys, but has one shared key $K_{aj}$ which can be used with anyone. $K_2$ means that B has no public and private keys, but has one shared key $K_{ab}$ which is intended to share with A. $K_3$ means that J has a public key $PK_J$ and a private key $SK_J$, but J has no share key. $K_4$ means that the attacker has no public and private keys, but has one shared key $K_i$ which can be used with anyone.

Let $Sch$ be $[1,1,1,1,2,2,2,2]$. In this schedule, "1" and "2" mean a complete execution of one protocol step in the first and second session, respectively. In the execution, both message sending and receiving in the step are performed. This means that a sender has sent a message, and a receiver has received the message. So the schedule is just the sequential execution of the first session and then the

second session. Consider another schedule *[1,2,2,1,1,2,2,1]*. This schedule is a non-sequential but concurrent execution which corresponds to the man-in-the-middle attack [30]. The man-in-the-middle attack means that the attacker situates in the middle between two sessions, and replays messages between them. The figure 1 illustrates the flow of messages for the man-in-the-middle attack in TMN where the first session is between A and *In*, and the second session is between *In* and B. Thus, the schedule means an alternating execution between two sessions in that the first protocol step of the 1st session is executed first, and then two protocol steps of the 2nd session are executed, and so on.

Let *Tr* be *([$K_{ab}$,$K_{aj}$], [$K_i$], [$K_{ab}$], LTr)* and *LTr* be *[(1,1,A,J, ((B,{$K_{aj}$}PK-J),A)), (1,1,In,J,((X2,{$K_i$}PK-J),X1)), ... ]*. The attack trace *Tr* consists of a vulnerability event *([$K_{ab}$,$K_{aj}$] [$K_i$] [$K_{ab}$])*, and a list *LTr* of protocol traces which leads to the attack for the vulnerability event. As for a simple example, the list *LTr* of protocol traces is only partial and contains two steps only. The first protocol trace *(1,1,A,J, (B,{$K_{aj}$}PK-J),A))* means that in the first session and at the first protocol step, user *A* sends message *(B,{$K_{aj}$}PK-J),A* to server *J*. The second protocol trace *(1,1,In,J,(X2,({$K_i$}PK-J),X1))* means that in the first session and at the first protocol step, the attacker *In* intercepts the message sent in the first trace, and modifies it to *(X2,{$K_i$}PK-J),X1* which is delivered to *J*.

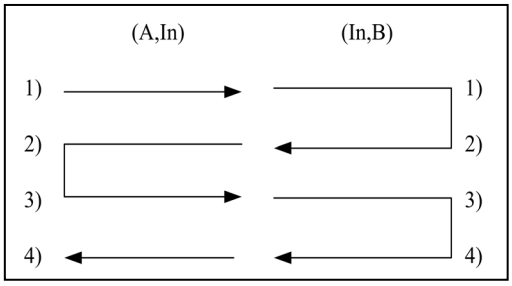


Figure 1: The flow of messages for the man-in-the-middle attack

To analyze the attacks in TMN, we consider only the schedule *[1,2,2,1,1,2,2,1]* which is clearly the execution of non-sequential but alternating sessions of the protocol. It suffices to consider only this schedule to illustrate the effectiveness of our CPN method. Other schedules can be considered too as different input parameters.

We consider four kinds of two concurrent sessions which are as follows.

1. *(1,A,B,J,($K_1$,$K_2$,$K_3$,$K_4$),_) & (2,In,In,J,($K_1$,$K_2$,$K_3$,$K_4$),_)*
2. *(1,A,In,J,($K_1$,$K_2$,$K_3$,$K_4$),_)& (2,In,B,J,($K_1$,$K_2$,$K_3$,$K_4$),_)*
3. *(1,In,B,J, ($K_1$,$K_2$,$K_3$,$K_4$),_)& (2,A,In,J,($K_1$,$K_2$,$K_3$,$K_4$),_)*
4. *(1,In,In,J, ($K_1$,$K_2$,$K_3$,$K_4$),_)& (2,A,B,J, ($K_1$,$K_2$,$K_3$,$K_4$),_)*

where $K_1$, $K_2$, $K_3$ and $K_4$ are discussed previously.

In session *(1,A,B,J,($K_1$,$K_2$,$K_3$,$K_4$),_)*, the attacker behaves as an external observer on the communication amongst A, B and J. In session *(1,A,In,J,($K_1$,$K_2$,$K_3$,$K_4$),_)*, the attacker impersonates a responder to user A and server J. In session *(1,In,In,J,($K_1$,$K_2$,$K_3$,$K_4$),_)*, the attacker impersonates both A and B to server J. Thus, there are three explicit roles of our attacker : an external observer and impersonators for initiator and responder. The role of server impersonation is implicitly enabled, but it is disabled in the session where both A and B are impersonated.

These four concurrent sessions are all possible sessions regarding to the goal of the user impersonation attack.

### 4.2.2. Our CPN graph model

Our CPN graph model for TMN extends the CPN graph model in [14] on many issues to provide the new methodology discussed in section 4.1. The CPN graph model consists of four levels: top, entity, sub-entity and control. The top level shows the interaction between all parties including the attacker. The entity level shows the detail of the overall behaviour of each party according to the protocol, and the sub-entity level shows the detail of a specific behaviour of a party. The control level controls the execution of the model according to an input schedule.

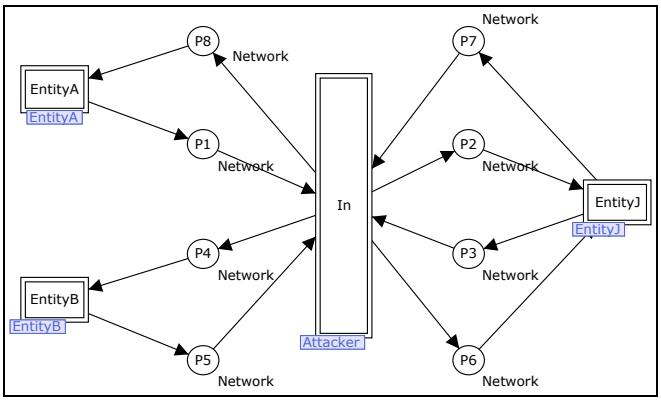


Figure 2: Top level

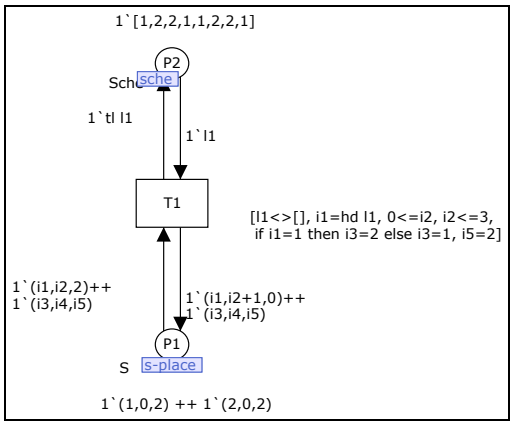


Figure 3: Entity A

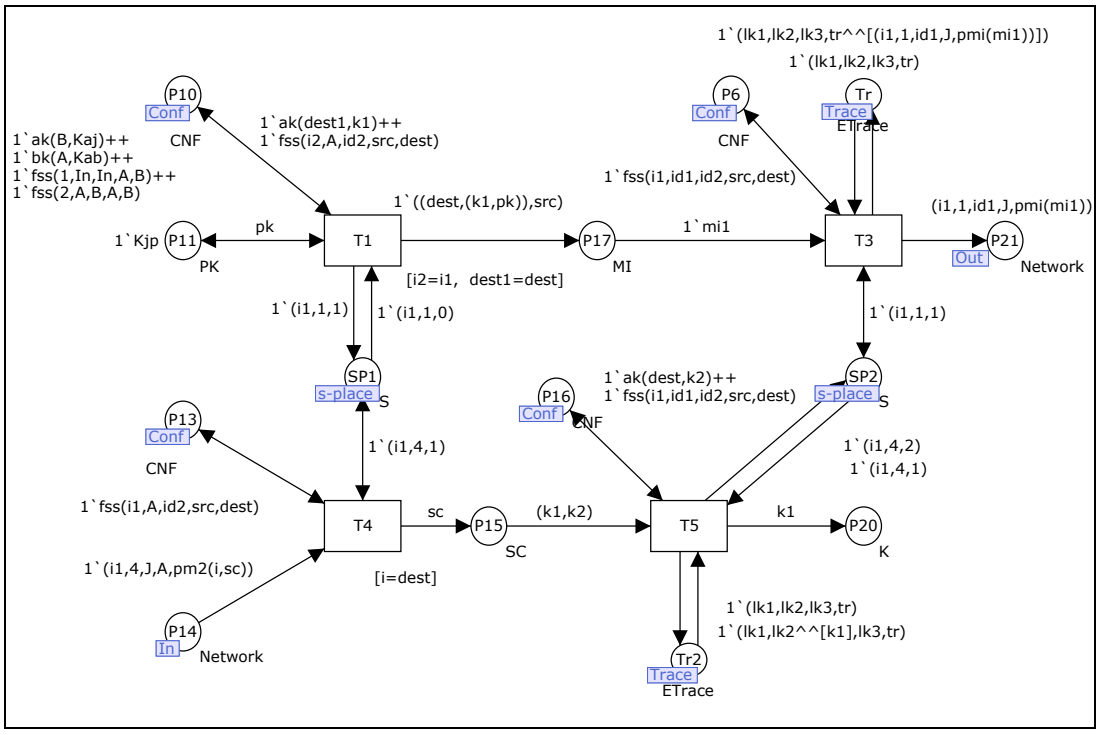


Figure 4: Entity A

Figure 2 shows the top level. There are 4 entities in our model which are A, B, J and attacker *In*. All messages that are exchanged between all users (A, B and J) pass through the attacker *In*. Figure 3 shows the control level. The input schedule *[1,2,2,1,1,2,2,1]* is given at the *Sche* place. The place *s-place* keeps session states where a session state consists of (*sid, sp, st*) where *sid* means the session identity, *sp* means the counter of protocol steps and *st* means states. There are three states which are 0 (ready), 1 (executing) and 2 (inactive or finished). For example, state (1,1,1) means that the session 1 is during the execution at the first protocol step. Initially, we have two session states: (1,0,2) and (2,0,2). Transition T1 activates the execution of a session at a time according to the schedule, and also increments the counter of protocol steps to be executed. As a result of the activation, the protocol step at the counter is executed.

Figure 4 shows the entity level for A. Transitions T1 and T3 are for creating the message sent at the first step of the protocol, whereas transitions T4 and T5 are for processing the message received at the last step. The place *Conf* stores a session configuration which consists of *fss, ak* and *bk* tokens where *fss(sid, i1, i2, i3, i4)* means that in session identity *sid, i1* and *i2* are identities of the actual initiator and actual responder, respectively, and *i3* and *i4* are identities that *i1* and *i2* use, respectively, in the message. Token *ak(id,k)* means that *k* is a shared key between A and *id*, and *bk(id,k)* describes a shared key *k* between B and *id* similarly. The place *trace* stores an attack trace when A sends a message to J. Transition T1 is enabled if session state is *(i1,1,0)* and A is the actual initiator in session *i1*. After the session *i1* is executed, its session state becomes *(i1,1,1)*. Transition T4 is to validate a received message, and T5 is to decrypt the received ciphertext. After the step 4 is terminated, its session state becomes *(i1,4,2)*.

#### 4.2.3. Queries

After a state space is computed for each configuration, we search for attack states in the state space. As discussed previously, attack states are characterized by vulnerability events. It is straightforward to construct queries to detect the combined vulnerability events. However, our method is that queries are applied to only terminal states or nodes in the state space in order to verify if certain conditions are met. The terminal states are states which do not have any further possible computation, and they mean states at the completion of the protocol execution. Note that while we consider terminal states, Al-Azzoni et. al. in [14] consider all states which are unnecessary and inefficient.

The following shows a query for the first basic vulnerability event.

```
val LeafNodes=ListDeadMarkings();
fun SecrecyViolation1(k:K) :
Node list
= PredNodes (LeafNodes,
            fn n => (cf(cK(k), Mark.SymDec'P3 1 n) > 0),
             NoLimit);
```

The *SecrecyViolation1*($K_{aj}$) produces a set of all terminal nodes in the state space where the key $K_{aj}$ is present at the attacker's database which is represented by the place number P3.

The following shows a query for the second case of the second basic vulnerability events.

```
fun SecrecyViolation2(k:K) :
Node list
= PredNodes (LeafNodes,
            fn n => (cf(k, Mark.EntityA'P20 1 n) > 0),
            NoLimit);
```

The *SecrecyViolation2*($K_{aj}$) produces a set of all terminal nodes in the state space where the key $K_{aj}$ is present at A's database after the completion of the protocol. Note that A's database is represented by the place number P20.

So, a query to detect the combined vulnerability events can be created by applying the intersection on the queries with appropriate keys. For example, the combined event 1 is obtained by the intersection on the queries *SecrecyViolation1*($K_{ab}$) and *SecrecyViolation2*($K_{ab}$). Note that we do

not need to query for B's commitment on a session key since B always commits on key $K_{ab}$ due to the protocol specification.

#### 4.2.4. Obtaining attack traces

After the attack states are found by using the queries, we extract attack traces which describe how the attacker carries out those attacks successfully step by step. Our method to extract attack traces is very efficient. Since such an attack trace is recorded into a global fusion place while the protocol execution proceeds step by step. Thus, an attack trace can be obtained from the global fusion place in an attack state immediately. Note that in other CPN approaches for security protocols, an expensive computation is required to find a path from an initial state to an attack state, and then to extract relevant information from the path to create an attack trace.

#### 4.2.5. Attack Classification

Usually, there can be a huge number of attack states found in a state space. In fact, we found 360 different attack states in a state space obtained from in the first configuration as shown in table 1.

After we obtain a large amount of the found attack traces, we analyze them manually first. We found that many of the attack traces share a similar pattern. In general, ciphertexts that are sent in all messages are essential to the attacks, but the identities of initiator and responder in the messages that are modified by the attacker are not important. However, the identities of initiator and responder in the messages that are sent from A in step 1, sent to B in step 2 and sent to A in step 4 are important. These identities are used by J for the key exchange function and by A and B for input validation. If we ignore unimportant parts of protocol messages in protocol traces, then we obtain a trace pattern. By considering trace patterns, the number of attacks is decreased tremendously as shown in table 1. Indeed, the process to find trace patterns is manual and protocol-dependent.

Then, we construct an attack pattern from similar attack traces. An attack pattern consists of a vulnerability event and a trace pattern for the event. The vulnerability event is the result of the attack, and the trace pattern is a list of minimal protocol traces which lead to the attack. An attack pattern is a general form of many attacks of the same kind. We argue that the attack pattern is more suitable for the protocol analysis than a detailed attack trace.

We develop an automated attack classification method to classify a huge amount of attack traces found by using attack patterns. First, given a current set of attack traces to be classified, we create a new attack pattern by taking the first attack trace in the set, and add the new pattern into a current set of known patterns. Then, by using a current set of known attack patterns, we filter out the known attack patterns from a current set of attack traces to be classified. And the process repeats again until there is no output set of attack traces to be classified.

#### 4.2.6. New Attacks

The table 1 shows the number of attacks states (and attack traces) and attack patterns found in each configuration. In table 1, *(A,B) (In,In)* means the first configuration of two concurrent session where A and B are in the first session, and *In(A)* and *In(B)* are in the second session. Also, *Tr* and *Pat* means the number of attack traces and attack patterns, respectively. Note that our method does not find attack states for the events 1 and 3. Also, all attack traces found from the second, third and fourth configurations are just some variant forms of those attack traces from the first configuration. Therefore, in the following we discuss only the attacks from the first configuration.

| **Configurations** | **Event 2** | | **Event 4** | | **Event 5** | |
|---|---|---|---|---|---|---|
| | **Tr** | **Pat** | **Tr** | **Pat** | **Tr** | **Pat** |
| 1. (A,B) (In,In) | 360 | 10 | 360 | 10 | 360 | 10 |
| 2. (A,In) (In,B) | 144 | 4 | 144 | 4 | 144 | 4 |
| 3. (In,B) (A,In) | 72 | 2 | 72 | 2 | 72 | 2 |
| 4. (A,B) (In,In) | 36 | 1 | 36 | 1 | 0 | 0 |

*Table 1:* Number of Attack Traces and Patterns

By using our new CPN method, we found two new attacks of the TMN protocol for multiple sessions of protocol execution. The first and the second new attack is in the category of the combined vulnerability event 4 and 5, respectively. Also, we found many interesting variant forms of Lowe and Roscoe's attack.

The first new attack is the combined vulnerability event 4 where the attacker learns the session key and A's secret, and fools A to commit to a fake key which is the attacker's secret $K_i$. The event 4 represented by $[K_{ab},K_{aj}][K_i][K_{ab}]$ leads to a kind of the man-in-the-middle attacks in that the attacker uses $K_i$ to learn the communication from A and use $K_{ab}$ to create a fake communication to B. As a result, the attacker learns the communication between A and B, but both A and B think that they communicate to each other. In the event 4, we found 10 attack patterns. Each attack pattern represents a group of similar attacks with a slight and unimportant difference. Due to the space limit, we discuss only some attack patterns only. A full detail of the attacks in TMN is described in [21]. The following shows the first attack pattern.

1) $A \rightarrow In(J) : (B, \{K_{aj}\}PK\text{-}J), A$
   $In(J) \rightarrow J : (X2, \{K_i\}PK\text{-}J), X1$
1') $In(A) \rightarrow J : (X4, \{K_i\}PK\text{-}J), X3$
2') $J \rightarrow In(B) : X3$
2) $J \rightarrow In(B) : X1$
   $In(B) \rightarrow B : A$
3) $B \rightarrow J : (X1, \{K_{ab}\}PK\text{-}J), X2$
3') $In(B) \rightarrow J : (X3, \{K_{aj}\}PK\text{-}J), X4$
4') $J \rightarrow In(A) : X4, E_{Ki}(K_{aj})$
4) $J \rightarrow In(A) : X2, E_{Ki}(K_{ab})$
   $In(A) \rightarrow A : B, E_{Kaj}(K_i)$

where $K_i$ is attacker's secret keys.

X1, X2, X3 and X4 stand for arbitrary identities that the attacker creates and uses in the messages during the attack. In step 1), the message that A sends to J is modified by the attacker. The original message is indicated by $A \rightarrow In(J)$, but the modified message by the attacker is indicated by $In(J) \rightarrow J$. Also, the messages at steps 2) and 4) are modified by the attacker.

In this attack pattern, the first message at the first session is modified to $\{K_i\}PK\text{-}J$, but it is replayed at the third step in the second session. Therefore, the attacker learns both $K_{ab}$ and $K_{aj}$, and the attacker sends $E_{Kaj}(K_i)$ at the fourth step in the first session.

The goal of the first attack is to learn keys $K_{aj}$ and $K_{ab}$ before the message at the last step is sent to A. And at the last step, $E_{Kaj}(K_i)$ is sent instead to fool A to commit to $K_i$. The attack can be understood by considering the keys that are encrypted by J's public keys at steps 1 and 3 in both sessions. Based on those keys, the message at step 4 contains the key at step 3 which is encrypted by key at step 1. And if the key at step 1 is known, then so is the key at step 3.

In the first session, $K_i$ and $K_{ab}$ are used in steps 1 and 3, respectively, and in the second session, $K_i$ and $K_{aj}$ are used in steps 1 and 3, respectively. These can be represented by the notation $(<K_i,K_{ab}>,<K_i,K_{aj}>)$ where $(P_1,P_2)$ means that $P_1$ and $P_2$ are information for the first and second sessions, respectively. Thus, the ciphertexts obtained at the step 4 in the first and second sessions are $E_{Ki}(K_{ab})$ and $E_{Ki}(K_{aj})$, respectively. So, the attacker obtains the target keys easily.

The second attack pattern which can be represented by $(<K_i,K_{aj}>,<K_i,K_{ab}>)$ is similar to the first attack pattern. The difference is on steps 3), 4), 3') and 4'). The different steps in the second attack pattern are shown as follows.

3) $B \rightarrow In(J) : (X1, \{K_{ab}\}PK\text{-}J), X2$
   $In(J) \rightarrow J : (X1, \{K_{aj}\}PK\text{-}J), X2$
3') $In(B) \rightarrow J : (X3, \{K_{ab}\}PK\text{-}J), X4$
4') $J \rightarrow In(A) : X4, E_{Ki}(K_{ab})$
4) $J \rightarrow In(A) : X2, E_{Ki}(K_{aj})$
   $In(A) \rightarrow A : B, E_{Kaj}(K_i)$

The third attack pattern which can be represented by $(<K_i,K_{ab}>,<K_{aj},K_i>)$ is shown as follows.

1) $A \rightarrow In(J) : (B, \{K_{aj}\}PK\text{-}J), A$
   $In(J) \rightarrow J : (X2, \{K_i\}PK\text{-}J), X1$
1') $In(A) \rightarrow J : (X4, \{K_{aj}\}PK\text{-}J), X3$
2') $J \rightarrow In(B) : X3$
2) $J \rightarrow In(B) : X1$
   $In(B) \rightarrow B : A$
3) $B \rightarrow J : (X1, \{K_{ab}\}PK\text{-}J), X2$
3') $In(B) \rightarrow J : (X3, \{K_i\}PK\text{-}J), X4$
4') $J \rightarrow In(A) : X4, E_{Kaj}(K_i)$
4) $J \rightarrow In(A) : X2, E_{Ki}(K_{ab})$
   $In(A) \rightarrow A : B, E_{Kaj}(K_i)$

The seven remaining attack patterns are $(<K_i,K_{ab}>,<K_{aj},K_{ab}>)$, $(<K_i,K_{aj}>,<K_{aj},K_{ab}>)$, $(<K_{aj},K_i>,<K_i,K_{ab}>)$, $(<K_{aj},K_{ab}>,<K_i,K_{ab}>)$, $(<K_{aj},K_{ab}>,<K_i,K_{aj}>)$, $(<K_{aj},K_{ab}>,<K_{aj},K_i>)$ and $(<K_{aj},K_i>,<K_{aj},K_{ab}>)$.

The second new attack is the combined vulnerability event 5 represented by $[K_{ab},K_{aj}][K_{aj}][K_{ab}]$. The event 5 leads to the similar kind of the man-in-the-middle attacks to the event 4, but here the attacker uses $K_{aj}$ to learn the communication from A. As a result, the attacker learns the communication between A and B, but both A and B think that they communicate to each other. In the event 5, we found 10 attack patterns. These attack patterns are similar to those patterns for the event 4 except that the last step of the first session is change to the following.

4) $J \rightarrow In(A) : X2,\ldots$
   $In(A) \rightarrow A : B, E_{Kaj}(K_{aj})$

Conceptually, the difference between the first new attack and the second new attack is that in the former, $E_{Kaj}(K_i)$ is sent to A in the last step, but in the latter, $E_{Kaj}(K_{aj})$ is sent to A in the last step. So, user A is fooled to commit on his own secret.

Finally, we also found many interesting variant forms of Lowe and Roscoe's attack where the server J cannot detect the replay attack in the two concurrent sessions. Note that in [29], Zhang and Liu detected only one variant form of this kind, and so they can detect less number of attacks than our approach. Due to space limit, the details of these variants are omitted here.

#### 4.2.7. Performance

The table 2 shows the computation time and the size of a state space from each configuration. In our experiment, the computation of the state space is executed on a PC with Intel Core2 Duo 2.33 Ghz with 2 GB of RAM.

| **Configurations** | **Nodes** | ***Arcs*** | ***Time (sec.)*** |
|---|---|---|---|
| 1. (A,B) (In,In) | 104,346 | 109,476 | 976 |
| 2. (A,In) (In,B) | 73,806 | 77,568 | 523 |
| 3. (In,B) (A,In) | 51,212 | 52,639 | 282 |
| 4. (A,B) (In,In) | 34,160 | 35,095 | 120 |

Table 2. Size and Time of the generated state spaces

At most, it takes 16 minutes to compute a state space to analyze the attack.

#### 4.2.8. Discussion

We verify the completeness of our method by hands and found that the attack patterns discovered for the events 4 and 5 are complete with respect to the assumptions stated in section 4.2.1. Our argument is as follows. Consider the attack patterns in event 4 only. Let $P$ be $(<\{K_i,K_{aj}\}, \{K_{ab},K_{aj},K_i\}>, <\{K_i,K_{aj}\}, \{K_{ab},K_{aj},K_i\}>)$ which uses a similar notation to the attack pattern representation. $P$ means that in the first session, only the ciphertexts of $K_i$ and $K_{aj}$ and the ciphertexts of $K_{ab}$, $K_{aj}$ and $K_i$ can be sent or replayed at step 1 and at step 3, respectively, and similarly in the second session. Clearly, these are true. And it is easy to verify that all of our attack patterns are just all enumerations from $P$ which lead to the attacks. Note that each enumeration is constructed by selecting each possible key to produce a ciphertext in each session.

Exploring all possible alternating executions within a state space is expensive and causes a huge state space. In our experiment, it takes more than 6 days to compute such a state space. Moreover, some alternating executions may be unnecessary but redundant. For example, schedule *[2,2,2,1,1,1,2,1]* is similar to schedule *[2,2,2,2,1,1,1,1]* regarding to attacks found since in the former, the information obtained from the three steps in session 1 is not useful for any replay attack on session 2. Note that step 4 contains a shared-key ciphertext, but steps 1 and 3 contain public-key ciphertexts. Similarly, schedules *[1,1,2,2,1,2,1,2]* is similar to *[1,2,1,2,1,2,1,2]* since step 2 in session 1 contains only plaintext which is already known by an attacker, and thus no information at step 2 in session 1 is useful for any replay attack on session 2.

### 4.3. Our CPN Analysis for Micali's ECS1 Protocol

In this section, we discuss the analysis of ECS1 by using our CPN method. In fact, our CPN framework for the analysis of ECS1 is similar to the framework for TMN discussed previously. So, we discuss only the difference between them. Also, some new attacks on ECS1 will be discussed in section 4.3.2.

#### 4.3.1. Our CPN framework for Micali's ECS1 Protocol

The assumptions of the protocol execution for ECS1 are similar to those assumptions in definition 1 except for assumption 5. For ECS1, the same initiator and responder may participate in more than one session.

We assume two kinds of attackers which are *I* and *Ar*. The attacker *I* is exactly the same as the attacker *In* discussed in section 4.2.1. However, *Ar* is a different kind of attackers, and it is considered as a conspired user with attacker *I*. In fact, *Ar* is a participating user in the system who has all the abilities discussed in definition 2, except for the assumption 1. An attacker who has the ability in the assumption 1 can behave as an external observer who tries to manipulate messages between users in a session. Note that the external observer is not a user in the session. But, *Ar* is a malicious user who participates in a session and conspires with an attacker by sharing some information. These two attackers collaborate to cheat a user. Note that one attack found by Bao et. al. [28] involves these two kinds of attackers.

Attack states in ECS1 are characterized by vulnerability events. There is one vulnerability event in ECS1 protocol which is unfair states. An unfair state means that one party, who is either initiator or responder, gets another party commitment, but the latter does not get the former commitment. In fact, there are two kinds of unfair states where the initiator has an advantage and the responder has an advantage, respectively.

**Definition 9 :** The vulnerability events in ECS1

There are two unfair states.

1) The initiator has the responder's commitment, but the responder does not have the initiator's commitment. *(AgainAdv)*
2) The responder has the initiator's commitment, but the initiator does not have the responder's commitment. (*BgainAdv*)

First, we compute two basic vulnerability events where the responder has initiator's commitment (*ACommit*) and the initiator has responder's commitment (*BCommit*). Then, fair states (*BothCommit*) are computed by ($ACommit \cap BCommit$). Finally, the two main vulnerability events

can be computed by *AgainAdv = (BCommit – BothCommit)* and *BgainAdv = (ACommit – BothCommit)*.

The queries to compute *ACommit* and *BCommit* are straightforward. For example, to obtain *ACommit*, we write a query to find a set of all terminal nodes in the state space where initiator's commitment, which are $SIG_A(C,Z)$ and M, are present at responder's database. Also other queries can be created easily.

#### 4.3.2. New attacks

We found one new single-session attack of Micali's ECS1, two new multi-session attacks in Micali's ECS1 and three new attacks of Bao's modified version of ECS1. Due to the space limit, we discuss only some new attacks in multiple sessions.

In the following, (1), (2), (3), … describe protocol steps in the first session but (1'), (2'), (3'), … describe steps in the second session.

We describe two new attacks in the original ECS1 protocol. In the first attack, a malicious responder gains an advantage over a well-behaved initiator if random *M* is reused in the two sessions. In the second attack, an initiator gains an advantage over a well-behaved responder if the same contract is signed in the two sessions.

In the first attack, a well-behaved initiator A communicates with the same malicious responder *I* in two sessions. The same random *M* is used in both sessions but different contracts are used in the two sessions. The attacker (*I*) simply ignores to participate in one session. Since the random M from another session can be used to construct the same Z, attacker *I* has enough information to show that A has committed to *I* in both sessions. The attack is shown as follows.

*1) $A \rightarrow I : SIG_A(C1,Z1)$ where $Z1=ENC_{TTP}(A,I,ma1)$*
*1') $A \rightarrow I : SIG_A(C2,Z1)$ where $Z2=ENC_{TTP}(A,I,ma2)$*
*2') $I \rightarrow A$ : Nothing*
*2) $I \rightarrow A : SIG_I(C1,Z1), SIG_I(Z1)$*
*3) $A \rightarrow I$ : ma1*

Note that step 2') means that I does not send any message to A, and thus the second session is aborted.

In the second attack, a well-behaved initiator communicates with a malicious responder *I* in one session but communicates with a well-behaved responder B in another session. The same contract is used in the two sessions. The malicious responder *I* replays A's signature for session with *I* to the session with B. Due to the duplicate contract, A has B's commitment. However, B does not have A's commitment since A's signature is replayed, and thus Z is not correct. Note that TTP cannot resolve the dispute due to the incorrect Z. The attack is shown as follows.

1) *$A \rightarrow I : SIG_A(C1,Z1)$ where $Z1=ENC_{TTP}(A,I,ma1)$*
*1') $A \rightarrow I(B) : SIG_A(C1,Z2)$ where $Z2=ENC_{TTP}(A,B,ma2)$*
*$I(A) \rightarrow B : SIG_A(C1,Z1)$*
*2') $B \rightarrow A : SIG_B(C1,Z1), SIG_B(Z1)$*
*2) $I \rightarrow A$ : Nothing*
*3') $A \rightarrow B$ : ma2*
*4') $B \rightarrow TTP$ : A, B, $SIG_B(C1,Z1), SIG_B(Z1)$*
*5a') $TTP \rightarrow A$ : Error*
*5b') $TTP \rightarrow B$ : Error*

Since *I* does not send any message at step 2) in the first session, the session is ignored. In steps 5a') and 5b'), an error occurs due to the incorrect Z. Thus, no message is sent from TTP according to Micali's ECS1 protocol.

We discuss one new attack in Bao's modified ECS1 protocol only. In the attack, a well-behaved initiator communicates with malicious responder *I* in one session, but the malicious responder conspires with *Ar* in another session. As a result of the attack, a malicious responder gets a well-behaved initiator's commitment, but the initiator obtains *Ar*'s commitment instead. After *I*

receives A's signature in step 1, *I* forward A's signature to *Ar* and *Ar* requests TTP to resolve a dispute by replaying *Ar*'s signature. Thus, A will obtain *Ar*'s commitment instead, but *I* will obtain A's commitment. *I* gets A's commitment because TTP is fooled to issue random M to *I* by the help of *Ar*. The attack is shown as follows.

*1) $A \rightarrow I : SIG_A(C1,Z1)$ where $Z1 = ENC_{TTP}(A,I,ma1)$*
*In the second session, I sends $SIG_A(C1,Z1)$ to Ar secretly.*
*2) $I \rightarrow A$ : Nothing*
*4') $Ar \rightarrow TTP : A, Ar, SIG_{Ar}(C1,Z1), SIG_{Ar}(Z1)$*
*5') $TTP \rightarrow A : SIG_{Ar}(C1,Z1), SIG_{Ar}(Z1)$*
*6') $TTP \rightarrow Ar$ : ma1*
*In the second session, Ar sends ma1 to I secretly.*

#### 4.3.3. Performance

The table 3 shows some examples of configurations and their computation results. Each row of the table shows a simplified form of a configuration of a session. For example, *(A,I,c1,ma1)* means that in the session *A* and *I* are initiator and responder, respectively, and *c1* is a contract and *ma1* is the random M. Note that "_" means any value. Also, "All States" means the number of all states in a state space and "Attack states" means the number of attack states found in the state space. "Times" means the amount of times in seconds that are used to generate each state space.

| Configurations of two sessions | All States | Times (in seconds) | Attack States |
|---|---|---|---|
| (Ar,I,_,mi1)&(I,B,c1,mi1) | 146,318 | 3,163 | 1,900 |
| (Ar,I,_,mi1)&(A,B,c1,ma1) | 78,594 | 1,806 | 2,048 |
| (I,Ar,_,mi1)&(A,I,c1,ma1) | 91,538 | 2,093 | 330 |
| (I,Ar,_,mi1)&(I,B,c1,mi1) | 339,918 | 13,200 | 2,952 |
| (I,Ar,_,mi1)&(I,B,c1,mi2) | 647,918 | 34,370 | 7,032 |
| (I,Ar,_,mi1)&(A,B,c1,m1) | 51,434 | 1,188 | 1,104 |
| (A,I,c1,ma1)&(A,I,c2,ma1) | 2,021 | 25 | 28 |
| (A,I,c1,ma1)&(A,I,c2,ma2) | 5,689 | 77 | 0 |
| (A,I,c1,ma1)&(I,B,c1,mi1) | 7,004 | 85 | 108 |

Table 3: Some results of the state space

Note that at some configuration, it requires 9 hours to compute the state space.

### 4.4. Discussion

Even though our CPN method offers many advantages, there is some disadvantage also. The size of each state in the computed state space is large since an attack trace is embedded into each state. As a future work, we aim to improve our method to overcome this disadvantage. Also, we aim to apply our new method to analyze other kinds of cryptographic protocols and to analyze other kinds of attacks.

## 5. Conclusion

In this paper, we propose a new CPN methodology for the security analysis of cryptographic protocols. Our approach offers a simple but effective way to analyze multiple sessions of protocol execution. We argue that our new CPN methodology improves on all existing CPN and Petri net methods for security protocols on several issues. In particular, our CPN method is the first CPN method which offers a security analysis methodology of multiple concurrent sessions of protocol execution. Furthermore, it offers a systematic method to analyze attacks in protocols. In particular, our method offers the decomposition and multi-session scheduling for the computation of state spaces, the intuitive approach to characterize attack states, the efficient way to extract attack traces and the systematic way to classify a large amount of attack traces. Also, it can detect more attacks with a better efficiency than all existing CPN methods for security protocols.

To demonstrate the practical uses of our approach, we apply our methodology to two case studies which are Micali's contract signing protocol ECS1 and TMN authenticated key exchange protocol. Surprisingly, we found many attacks in the two protocols. For ECS1, we found two new attacks in multiple sessions of protocol execution of the original ECS1, and three new attacks in Bao's modified ECS1. For TMN protocol, we found two new attacks in multiple sessions of protocol execution. In fact, the new attacks that we found in TMN protocol are quite surprisingly since TMN have been analyzed quite extensively.

## 6. Acknowledgement

We would like to thanks anonymous reviewers for their helpful and constructive comments. The first author would like to acknowledge a financial support from National Research Council of Thailand.

## 7. References

[1] J. Clark and J. Jacob, A survey on Authentication Protocols, *Research report*, University of York, 1997. (http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz)

[2] C. Meadows, Formal Verification of Cryptographic Protocols: A Survey, Advances in Cryptology - Asiacrypt '94, LNSC 917, Springer-Verlag, 1995

[3] C. Meadows, Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends, *IEEE Journal on Selected Areas in Communications*, 21(1), pp. 44-54, 2003.

[4] Ulrike Meyer, Susanne Wetzel, A man-in-the-middle attack on UMTS, In Proceedings of the 3rd ACM workshop on Wireless security, 2004, pp. 90-97

[5] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, Christopher Walstad: Breaking and fixing public-key Kerberos. *Information and Computation*, 206(2-4): 402-424, 2008

[6] Paul F. Syverson, A Taxonomy of Replay Attacks, In proceedings of the 7th IEEE Computer Security Foundations Workshop (CSFW), pp. 187-191, 1994.

[7] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEE, 77(4):541-580, 1989.

[8] C. Morton, L. Robart and S. Tavares, Decomposition Techniques for Cryptographic Protocol Analysis, *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, Canada, 1994.

[9] A. Basyouni and S. Tavares, New Approach to Cryptographic Protocol Analysis using Coloured Petri Nets, *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, Canada, 1997.

[10] G. Lee and J.Lee, Petri Net Based Models for Specification and Analysis of Cryptographic Protocols, *The Journal of Systems and Software*, 37:141-159, 1997.

[11] S. Lim, J. Ko, E. Jun and G. Lee, Specification and analysis of n-way key recovery system by Extended Cryptographic Timed Petri Net, *The Journal of Systems and Software*, 58:93-106, 2001.

[12] Suratose Tritilanunt, Colin Boyd, Ernest Foo, Juan Manuel González Nieto: Using Coloured Petri Nets to Simulate DoS-resistant Protocols. In *proceedings of Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 261-280. University of Aarhus, Denmark, October 2006..

[13] W. Dresp, Security Analysis of the Secure Authentication Protocol by Means of Coloured Petri Nets, *Proceeding of 9th IFIP Communications and Multimedia Security*, 2005.

[14] Al-Azzoni, I., Down, D.G., and Khedri, R., "Modeling and Verification of Cryptographic Protocols Using Coloured Petri Nets and Design/CPN", *Nordic Journal of Computing*, 12(3): 201-228, 2005.

[15] F. Crazzolara and G. Winskel, Events in Security Protocols, *Proceedings of the 8th ACM Conference on Computer and Communication Security*, USA, 2001.

[16] K. Jensen, Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Vol.1. Monographs in Theoretical Computer Science, Springer-Verlag, 1997

[17] K. Jensen, L.M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems", *International Journal on Software Tools for Technology Transfer*, 9(3): 213-254, 2007.

[18] Y. Permpoontanalarp and W. Rujiwattanaphong, "An Improved Method to Analyze Cryptographic Protocols by using Coloured Petri Nets", *Proceedings of 1st Joint International Conference on Information Communication Technology*, Laos, 2007.

[19] P. Sornkhom and Y. Permpoontanalarp, Security Analysis of Micali's Fair Contract Signing Protocol by Using Coloured Petri Nets, *Proceedings of the 9th ACIS International Conference on Software*

*Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Thailand, IEEE press, 2008.

[20] P. Sornkhom and Y. Permpoontanalarp, Security Analysis of Micali's Fair Contract Signing Protocol by Using Coloured Petri Nets : Multi-session case, *In Proceedings of the 5th International Workshop on Security in Systems and Networks*, Italy, IEEE press, 2009.

[21] Y. Permpoontanalarp, Security Analysis of the TMN protocol by using Coloured Petri Nets, Technical Report, King Mongkut's University of Technology Thonburi, Bangkok, Thailand, 2009.

[22] S. Micali, "Simple and Fast Optimistic Protocols for Fair Electronics Exchange", *Proceedings of 21st Symposium on Principles of Distributed Computing.* USA, pp. 12-19, 2003.

[23] M. Tatebayashi, N. Matsuzaki, and D. Newman, Key Distribution Protocol for Digital Mobile Communication Systems, *In CRYPTO-89*, Springer-Verlag, 1990.

[24] R. Kemmerer, C. Meadows and J. Millen, Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2), 1994.

[25] J. Mitchell, M. Mitchell and U. Stern, Automated analysis of cryptographic protocols using Murϕ, *In IEEE Symposium on Security and privacy*, 1997.

[26] G. Lowe and B. Roscoe, Using CSP to Detect Errors in the TMN Protocol, *IEEE Transactions on Software Engineering*, 23(10), 1997.

[27] Y. Zhang and X. Liu, An approach to the formal analysis of TMN protocol, *Progress on Cryptography: 25 years of Cryptography in China*, Springer-Verlag, 2004.

[28] F. Bao, G. Wang, J. Zhou, and Z. Zhu, "Analysis and Improvement of Micali's Fair Contract Signing Protocol", *Proceedings of The 9th Australasian Conference on Information Security and Privacy*. Australia, 2004: 176-187.

[29] Yuqing Zhang, Zhiling Wang, Bo Yang, The Running-Mode Analysis of Two-Party Optimistic Fair Exchange Protocols, *International Conference on Computational Intelligence and Security*, Springer Verlag, 2005.

[30] Gavin Lowe, An Attack on the Needham-Schroeder Public-Key Authentication Protocol, *Information Processing Letters*, 56(3): 131-133 (1995)

[31] Thomas Y. C. Woo, Simon S. Lam: A Lesson on Authentication Protocol Design. *Operating Systems Review* 28(3): 24-37 (1994)

[32] R. Chadha, M. Konovich and A. Scedrov, Inductive Methods and Contract-Signing Protocols, *Proceedings of 8th ACM Conference on Computer and Communications Security*, 2001.

[33] V. Shmatikov and J.C. Mitchell, Finite-State Analysis of Two Contract Signing Protocols, *Theoretical Computer Science*, 283:419-450, June 2002.

[34] S. Gürgens and C. Rudolph, Security analysis of efficient (Un-)fair non-repudiation protocols, *Formal Aspect of Computing*, 17(3): 260-276, 2005.

[35] D. Dolev, A. Yao, On the security of public key protocols, *IEEE Transactions on Information Theory*, 29(2): 198-207, 1983.